**Classifiying MRI Data as Demented or Nondemented**

**Alexander Hay**

**Goal**

To build a model using Deep Learning and Tensorflow that can accurately classify T1-weighted MRI scans as demented or nondemented.

**Introduction**

Dementia is a syndrome – usually of a chronic or progressive nature – in which there is deterioration in cognitive function (i.e. the ability to process thought) beyond what might be expected from normal ageing. It can result from a number of a variety of diseases and injuries, such as Alzheimer's disease or stroke. Around 50 million people worldwide have dementia, and every year, there are nearly 10 million new cases. The total number of people with dementia is projected to reach 82 million in 2030 and 152 million in 2050. Although we don't yet have a cure for dementia, early diagnosis allows optimal management of patients. Doctors can anticipate potential problems, and prescribe medications or lifestyle changes that can improve patients' quality of life. (WHO, https://www.who.int/news-room/fact-sheets/detail/dementia)

Despite the importance of early diagnosis, tests such as brain scans are time-intensive and costly. An MRI of the brain takes about 45 minutes and can cost $250-12,000 depending on the scan type and clinic location (https://www.cedars-sinai.org/programs/imaging-center/exams/mri/brain.html, https://affordablescan.com/blog/brain-mri-cost/Machine). Not to mention, currently, a radiologist needs to examine the image and provide a diagnosis. But what if there was a way to automate this process, reducing the time and costs required, and allowing more people to be diagnosed early? Deep learning and Tensorflow provide a promising solution to this problem. Models have already been developed that can accurately diagnose diabetic retinopathy, pneumonia, and Alzheimer's Disease from medical image data, but there is still room for improvement. Therefore, in this project I make a first attempt to solve this dementia classification problem. I have applied what I have learned in BIOF399 and attempted to build a model in Tensorflow that can classify T1-weighted MRI scans as demented or nondemented.

**Data**

I am using the OASIS-2 dataset, which consists of a longitudinal collection of 150 subjects aged 60 to 96. Each subject was scanned on two or more visits, separated by at least one year for a total of 373 imaging sessions. For each subject, 3 or 4 individual T1-weighted MRI scans obtained in single scan sessions are included. The subjects are all right-handed and include both men and women. 72 of the subjects were characterized as nondemented throughout the study. 64 of the included subjects were characterized as demented at the time of their initial visits and remained so for subsequent scans, including 51 individuals with mild to moderate Alzheimer’s disease. Another 14 subjects were characterized as nondemented at the time of their initial visit and were subsequently characterized as demented at a later visit (Marcus et al. 2010).

Source: OASIS: Longitudinal: Principal Investigators: D. Marcus, R, Buckner, J. Csernansky, J. Morris; P50 AG05681, P01 AG03991, P01 AG026276, R01 AG021910, P20 MH071616, U24 RR021382 (https://www.oasis-brains.org/)

**Accessing and filtering the data**

I downloaded the data using the `download\_data.py` script in this repository. The resulting dataset was nearly 46 GB, which was quite large given my limited computing power. I filtered and compressed the data using `filter\_data.py` to make the data more manageable. The resulting file, `filtered\_data.zip`, was just under 2 GB, small enough to be uploaded to my Google Drive to be accessed in Google Colab. To filter the data, I did the following things:

1. I removed all the 'converted' subjects, because it was unclear at which point they went from dementia-free to having dementia.
2. For each subject, I kept just the first scan from their first visit. This further cut down the file size.

**Loading data and preprocessing**

I loaded the MRI data using the nibabel package. I normalized the image to zero mean and unit variance, and resized it to a smaller numpy array. Normalization helps to account for noise, artifacts and uneven sensitivity of the scanner within the tissue region. I resized the image simply because it allowed for faster training and processing speed.

I created lists of filepaths for my training and validation data. I did so via a 70/30 split. The classes were pretty well balanced, as there were 72 nondemented scans and 64 demented scans. Because all of these images could not be stored in memory at once, I had to create a python generator function to continuously load data while the model was training. This function was used in a `tf.data.Dataset` object, which was shuffled upon loading the data and fed into the model with batch sizes of 2.

**Visualizing data**

I created functions that showed a single slice of the MRI, and a montage of slices. Here is an example:

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application

Description automatically generated

**Define model**

I created a model based on one by Zunair et al. 2020 (https://arxiv.org/abs/2007.13224 ). It is predicated on the idea that a 3D CNN should perform better than a 2D CNN on volumetric data, as there is important information in the depth dimension. A 2D CNN independently deals with individual slices and therefore discards the depth information. However, the problem with feeding into a 3D CNN is that it is much more memory-intensive. That is why in the data processing step I resized my volume from `(256, 256, 128)` to `(128, 128, 64)`. This downsampling allowed me to leverage the 3D information of the image without overwhelming my computer.

My model consisted of 4 3D convolutional layers, each followed by a max pooling and batch normalization step. The convolutions contained 64, 64, 128, and 256 filters respectively, and each had a kernel size of 3 and used a rectified linear activation function. I finished with a global average pooling layer, then a dense layer with 512 units, then a dropout layer to prevent overfitting. My output layer was a 1-unit dense layer with a sigmoid activation function, which is often used in binary classification tasks.

Table

Description automatically generated

**Model training**

I decided to use the `'adam'` optimizer, which has proven to be effective in many image classification problems, such as MNIST digit recognition. However, I implemented a slight twist based on the recommendation of Zunair et al., in that I set a conservative initial learning rate with an exponential decay. I used binary cross-entropy as my loss function since it is a binary classification task. I used accuracy as my metric, which is pretty standard. I set a callback to stop training if the validation loss stopped improving, and a callback to save the best performing model.

**Results: Model performance**

The model did not perform very well on the validation data. After 30 epochs, the validation loss stopped improving. The validation accuracy on the 30th epoch was actually just over 50%, but the best accuracy I achieved throughout training was 69%. Because that was the best performing model, that was the one that was saved. Here is a visualization of the model training.

Graphical user interface, application

Description automatically generated

*Important to note:* the training accuracy was actually quite high by the end, which is a sign of overfitting.

When I loaded that model and asked it to make predictions on single scans in the validation set, it was successful on the two examples that I selected.

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Discussion**

In the end, the model did not perform very well on the validation data. There are a few explanations for that.

1. The model was trained on very few data points. There were only 94 scans in the training set and 42 scans in the validation set. After 30 epochs, the model was overfitted to the training data and needed to be exposed to many more scans to become accurate on the validation set. However, as it was, the model training already took a very long time. 30 epochs of training took about 2 hours for me. This was because I had to use a generator function to load the data; I had to continously load scans throughout training rather than just access them in memory. One solution to this problem is to create a tf.records dataset, but I simply did not have enough time to do that. A tf.records dataset allows you to train much faster because you get better use out of your GPU; Tensorflow takes care of loading more data while your GPU is crunching your numbers. As I work to improve the model, that will be my first priority.
2. The data itself was flawed. Although I normalized the voxel intensities, I did not do spatial normalization. Therefore, the images probably had all sorts of different orientations. Some methods to counteract that are accounting for voxel spacing by resampling to an isotropic resolution, or registering the images to the same space. I could also augment the dataset, adding in noise, rotations, or translations to existing images to expose the model to more examples during training. That would likely decrease overfitting.
3. Finally, my model could have been flawed. Because so much time was spent preparing the data and training the model, I didn't get a chance to play around with some of the model parameters. Perhaps adding or removing layers, changing the number of filters or units, adjusting the amount of dropout, or changing the loss function, etc. could have improved performance. For example, I would like to try using the `'adam'` optimizer without setting a exponential decay learning rate, as that has been shown to work on many problems. I would also like to adjust the dropout, as my model was clearly overfitted to the training data. However, although these are all valid considerations, I think that first I should look to fix the data before the model.

**Conclusion**

It was good challenge to work with such a large dataset, and I learned a lot. To summarize:

* I set out to build a model in Tensorflow that could accurately classify T1-weighted MRI scans as demented or nondemented.
* My model did not perform very well, likely because there was not enough data, the data was flawed, and the model was not optimized.
* I would like to improve performance by creating a tf.records dataset, allowing the model to see more data since it will be processed faster, and by tweaking the data and model.

**References**

*Dataset:*

OASIS: Longitudinal: Principal Investigators: D. Marcus, R, Buckner, J. Csernansky, J. Morris; P50 AG05681, P01 AG03991, P01 AG026276, R01 AG021910, P20 MH071616, U24 RR021382 (https://www.oasis-brains.org/)

*Model Inspiration:*

Zunair, Hasib, et al. "Uniformizing Techniques to Process CT Scans with 3D CNNs for Tuberculosis Prediction." International Workshop on PRedictive Intelligence In MEdicine. Springer, Cham, 2020.

*Other sources:*

* WHO (https://www.who.int/news-room/fact-sheets/detail/dementia)
* Cedars Sinai (https://www.cedars-sinai.org/programs/imaging-center/exams/mri/brain.html)
* AffordableScan (https://affordablescan.com/blog/brain-mri-cost/Machine)